A Generalized, Enterprise-Level Systems Development Process Framework for Systems Analysis and Design Education

Heikki Topi
Bentley University
Waltham, MA 02452, USA
htopi@bentley.edu

Gary Spurrier
University of Alabama
Tuscaloosa, AL 35487, USA
gspurrier@cba.ua.edu

ABSTRACT

Current academic and industry discussions regarding systems development project approaches increasingly focus on agile development and/or DevOps, as these approaches are seen as more modern, streamlined, flexible, and, therefore, effective as compared to traditional plan-driven approaches. This extends to the current pedagogy for teaching systems analysis and design (SA&D). However, overemphasizing agile and DevOps neglects broader dimensions that are essential for planning and executing enterprise-level systems projects. Thus, a dilemma may arise: do we teach agile and DevOps techniques that may be inadequate for enterprise-level projects or do we teach the wider range of plan-driven skills and techniques that may conflict with the tenets and benefits of agile and DevOps? In this paper, we advocate for resolving this dilemma by adopting a generalized process framework that both fully supports enterprise-level projects but can also be selectively scaled back toward increased agility for smaller, less complex projects. In its full realization, this framework combines extensive project planning and up-front requirements with iterative delivery – an increasingly popular approach today for enterprise projects. In scaling back toward agile, the framework carefully accounts for system, environment, and team characteristics. Further, the model emphasizes issues frequently underemphasized by agile approaches, including the use of external software such as commercial-off-the-shelf (COTS), Software-as-a-Service (SaaS), and open source products and components; the need for business-oriented project planning and justification; and support for change management to ensure successful system adoption. The framework thereby flexibly accommodates the full range of activities that software projects must support to be successful.

Keywords: System development life cycle (SDLC), Agile, Enterprise systems development, Pedagogy, Systems analysis & design

1. INTRODUCTION

Systems analysis and design (SA&D) has been a central element of information systems education from the early days of the discipline. For example, the first curriculum recommendation for undergraduate degree programs in information systems (Couger, 1973) includes 2 related courses (out of 11) in this space, one called Information Systems Analysis and the other one System Design and Implementation. Likewise, Nunamaker, Couger, and Davis (1982) include a two-course sequence that "covers the application system development process" (p. 798), consisting of Information Analysis and Systems Design Process. In practice, these two courses provide a solid introduction to systems analysis and design plus the management of the software development process. Jumping almost 30 years forward to the latest curriculum recommendations, IS2010 (Topi et al., 2010) includes systems analysis and design as one of its seven core courses; in addition, IT project management is another core course that is closely related to SA&D. In the same way, one of the core competency areas in the graduate level curriculum recommendation MSIS 2016 (Topi et al., 2017) is systems development and deployment, and another one – innovation, organizational change, and entrepreneurship – is closely connected with it. Together, these areas cover capabilities typically associated with systems analysis and design and project management. Throughout its history, our discipline has consistently recognized the importance of SA&D and project management as core requirements at the same level as data(base) management; IT infrastructure; and IT policy, strategy, and management. The names of these key areas may

have varied over time, but the core elements have stayed surprisingly constant.

The IS discipline needs to continually ask whether or not the topics covered and competencies enabled in our core curricula are aligned with the long-term needs of the organizations hiring our students, the foundation that our graduates need to build for lifelong learning, and the expectations associated with the latest technological and methodological fashions. In recent years, issues surrounding this question have been sharpened by both academic and industry discussions regarding systems development methodologies and approaches that increasingly focus on replacing plan-driven methodologies – such as the traditional Systems Development Life Cycle (SDLC) or "waterfall" approach – with agile methods and/or DevOps (the latter defined as the extension of general agile principles to continuous integration and deployment; Gruver and Mouser, 2015). This is true both for the small, relatively simple projects for which agile was originally targeted but also, increasingly, for large, complex projects applicable to the enterprise. This is evidenced by extensive literature reviews concerning agile development in MIS journals (Dybå and Dingsøyr, 2008; Chuang, Luor, and Lu, 2014), as well as extended treatments by key practitioner authors pertaining to scaling agile methods to large-scale projects (Leffingwell, 2007; Ambler and Lines, 2012; Gruver and Mouser, 2015; Larman and Vodde, 2017; Knaster and Leffingwell, 2018).

It is, however, premature to claim that agile and/or DevOps are the best choice under all circumstances. Building on our earlier and ongoing work in this area (Spurrier and Topi, 2017) and recognizing ways in which pure agile methods and DevOps are misaligned with the reality of the development of administrative enterprise-level systems, we encourage the academic IS community to reconsider a broad range of questions related to the fit between systems development project characteristics and systems development approaches and methodologies. We also believe that identifying and acknowledging the importance of the factors affecting this fit is essential for success in educating the next generation of SA&D professionals.

As such, overemphasizing agile and DevOps neglects broader dimensions that are essential for planning, executing, and delivering enterprise-level systems projects. Thus, a dilemma may arise: do we teach agile and DevOps techniques that may be inadequate for enterprise-level projects or do we teach the wider range of plan-driven skills and techniques that may conflict with the tenets and benefits of agile and DevOps?

In this paper, we advocate for resolving this dilemma by adopting a generalized process framework for SA&D education that both fully supports enterprise-level projects and can be selectively scaled back toward increased agility for smaller, less complex projects.

By enterprise-level systems development, we refer to projects exhibiting several key characteristics (building on Fowler, 2003, pp. 2-4):

- Scope of significant size and complexity
- Supporting a large number of users in a variety of user actor roles
- Providing mission critical functionality and frequently also being a key to meeting the host organization's strategic objectives
- Utilizing a large number of user interface screens and complex business logic, implying large code bases, often measured in millions of lines of code
- Utilizing persistent databases measured in millions of rows and multiple gigabytes of data (or potentially significantly larger – billions of rows of data and terabytes of data is not uncommon)

In its full realization to support such enterprise-level systems development, this framework combines extensive project planning and up-front requirements specification with iterative delivery. It, therefore, represents a hybrid approach combining aspects of plan-driven and agile approaches. The framework also includes key planning dimensions of systems deployment that are not typically emphasized in DevOps, such as creating business policies and procedures, training materials, user acceptance testing, and data preparation.

In scaling back toward agile, the framework carefully accounts for system, environment, and team characteristics. Further, the model emphasizes issues frequently neglected by agile approaches, including the use of external software such as commercial-off-the-shelf (COTS), Software-as-a-Service (SaaS), and open source products and components; the need for business-oriented project planning and justification; and support for change management to ensure successful system adoption. The framework thereby recognizes the impacts and outcomes that all software projects have on their corresponding organizations.

We believe that following this framework would equip students with a broad toolkit that they could readily adapt to be effective in a wide range of project situations. Further, this approach would help position the information systems discipline better in the context of other, technology-centric computing disciplines (particularly computer science, information technology, and software engineering) and develop the specific, distinctive strengths of our degree programs. Specifically, our approach transcends the traditional, narrowly conceived definition of "systems development" referring mostly to "software construction," meaning technical design, coding, and testing. Rather, in this context, we define systems development to refer to not only construction but also configuration and integration of third party products and components, as well as extensive upfront requirements and project planning and backend change management/deployment tasks.

At its core, we argue that it is more effective to assume a broader, enterprise-level perspective that can be selectively simplified for smaller, less complex projects than to teach a narrow set of agile techniques that then need to be expanded in ad hoc ways to support larger and more complex projects. For example, we advocate teaching formal requirements techniques such as business process modeling and domain modeling that can be simplified as appropriate, rather than assuming an agile-style "barely sufficient" approach (Boehm and Turner, 2004, p. 18; Rubin, 2013, p. 57). Similar arguments apply to project planning and execution techniques. As such, we emphasize that our proposed approach would not ignore agile methods and approaches to systems development. Indeed, enterprise systems

development techniques and processes based on the hybrid approach include all the techniques and processes of agile plus a whole series of additional techniques for managing requirements, projects, and organizational change. Additionally, in a world where the great majority of teams do software construction iteratively, there is no fundamental conflict between agile and hybrid approaches. The main dimension of variability is the degree of project planning and requirements performed prior to construction and configuration: low amounts in agile and high amounts in hybrid.

Thus, enterprise software development represents a superset of systems skillsets, including those related to agile software construction. As such, a student schooled in enterprise-level techniques can easily learn to scale down those techniques to the needs of small, simple projects. In contrast, a student schooled in only agile techniques would struggle to successfully scale up those techniques to the enterprise level.

We originally developed this process framework for pedagogical purposes to be used as an organizing structure for systems analysis and design courses. We have also found it to be valuable for identifying areas where current pedagogical practices in the context of SA&D are not aligned with organizational needs and practices.

2. PRINCIPLES UNDERLYING THE PROPOSED FRAMEWORK

This section presents the underlying principles and beliefs that form the foundation for the proposed framework. Our motivation to propose a new framework rises from the belief that no existing approach captures and enables all these essential factors simultaneously and that existing SA&D courses frequently do not guide students to consider these factors in a comprehensive, systematic, and, above all, adaptive manner.

2.1 Development Approach Selection

We believe that all SA&D courses should recognize that every systems development project, however small, deserves dedication of time and resources to identify a development approach that best fits the characteristics of that project. It is important to note that there is no single, best, one-size-fits-all approach; sometimes a relatively agile approach will work best, and, in other situations, a more plan-driven approach will be more appropriate. Further, the same organization may need to utilize different development methodologies across its portfolio of projects. This would be determined by multiple factors, including characteristics of the system, the environment, and the team.

When analyzing the factors determining the positioning of a project on the plan-driven <–-> agile spectrum, we are depending and building on the "home grounds" theory articulated by Barry Boehm and Richard Turner (Boehm and Turner, 2004) in which software, scope, organization, and team characteristics play determinative roles. For example, per that prior work by Boehm and Turner (2004) and our extensions of that model (Spurrier and Topi, 2017), Table 1 summarizes a broad series of dimensions that determine the optimal approach to any given software project. A project that exhibited all of the characteristics in the Agile Home Ground column would be best served by utilizing a highly agile approach. Conversely, a project that exhibited all of the characteristics in the Plan-Driven Home Ground would be best served by a highly plan-driven approach. Further, specifically in our proposal, we reiterate that hybrid projects utilize a combination of formal, upfront requirements and project planning, iterative

Characteristic	Agile Home Ground	Plan-Driven Home Ground
Overarching context: Industry/Organization Characteristics		
Goals & Values	• Rapid, responsive delivery of value	• Predictable, high assurance delivery
Industry	• Turbulent/rapidly evolving	• Stable/mature
Organization	• Agile organization valuing freedom/empowerment/chaos	• Plan-driven organization valuing policies/procedures/control
IT Team contends with: Project/Application Characteristics		
Customers/ Product Owners	• Customers/Product Owners: Few, dedicated, co-located	• Customers/Product Owners: Many, not dedicated, not co-located
Software Requirements	• Small/flexible scope • Low development interdependence • Low clarity • Low stability over time • Single project focused	• Large/fixed scope • High development interdependence • High clarity • High stability over time • Project portfolio/organization-focused
Software Application	• "Greenfield" or small code base • Non-strategic/non-mission-critical • Low security/safety risk • No need for intentional architecture	• Large code base and/or "legacy" app • Strategic/mission-critical • High security/safety risk • High need for intentional architecture
IT Team	• IT Team: Small, generalists, co-located, high-performing, stable/cohesive, using tacit/shared informal knowledge	• IT Team(s): Large, specialists, multiple locations and time zones, few assumptions regarding performance levels, unstable/new, needing documentation for knowledge transfer

Table 1. Agile versus Plan-driven Extended Home Grounds Model

construction, and carefully planned change management and deployment.

The key point is that systems development can and should take on a broad variety of different forms. Indeed few, if any, systems development projects will optimally fully align with a specific, "pure," predefined development approach. Particularly, there are relatively few situations where a pure agile or a pure plan-driven approach to systems development provides the best fit. Rather, most projects would exhibit a mix of these characteristics, implying an approach that is neither "pure agile" nor "pure plan-driven." It follows that the proposed model is based on the assumption that some permutation of the two "pure" approaches will generally be found to be optimal.

Consequently, following a specific approach simply because it is required by group culture (e.g., "we are agile") or company policy is unlikely to lead to an optimal outcome. Instead, a specific system development approach should be formulated based on the key characteristics of the project. Understanding the factors affecting the development approach selection and the process used for selecting the approach are key capabilities that SA&D courses should cover. Unfortunately, many current courses do not recognize the essential role of development approach selection in SA&D projects.

2.2 Internal Development vs. Packaged Software and Components

Agile approaches to systems development projects tend to implicitly assume that a given systems project will involve a high degree of new software construction. Indeed, some observers have pointed out that agile approaches and methodologies tend to be code- or developer-centric, focused on software construction (meaning technical design, coding, and testing) and, conversely, paying relatively little attention to key questions outside software construction (Turk, France, and Rumpe, 2005; Leffingwell, 2007; West et al., 2011). This focus on software construction means that there is a relative lack of focus on projects where utilizing and integrating packaged software and components is (or could be) a central issue. Only recently has the academic literature started to pay attention to these essential questions (Petersen et al., 2018), and current textbooks appear to pay only occasional and superficial attention to the issues related to the use of purchased or other externally-sourced software capabilities in systems projects. Furthermore, few practitioner agile software methodology books provide any significant treatment of evaluating, utilizing, configuring, or integrating with third-party systems or components (Leffingwell, 2007; Ambler and Lines, 2012; Gruver and Mouser, 2015; Larman and Vodde, 2017; Knaster and Leffingwell, 2018).

For teaching SA&D, this is a serious omission. Systems development projects today almost always involve the integration of new software capabilities with existing ones. New software capabilities can either be developed from scratch or acquired from external sources through a variety of mechanisms, such as Software-as-as-Service (SaaS), Commercial-Off-The-Shelf (COTS) systems, use of modular components, open source capabilities, etc.

Of course, the use of commercial products, open source application projects, and/or pre-packaged software components may move a given project away from software construction and toward a significant emphasis on configuration, with software construction often limited to supplementing pre-written code with "surround code" extending its user interfaces and "glue code" integrating various systems and components with each other. This recognition of the importance and richness of a broader definition of "systems development" needs to be supported in our practice and teaching frameworks. Again, a key point is that enterprise-level systems projects take on a broad variety of different forms, and the extent to which new software development is part of it varies significantly.

We further note that the fundamental question of determining the extent to which a given systems project should depend on internal development versus external software capabilities implies the need to provide process support for evaluating sourcing approaches and selecting third-party products and components. This "build versus buy" decision is a familiar one in the general arena of defining and evaluating investment decisions, especially at the enterprise level. However, it is also one that is oddly absent from most of the agile practitioner literature and textbooks cited above. Thus, this is another key issue arguing for adopting and teaching a systems project approach that starts with the broader, enterprise-level perspective.

2.3 Meaningful Estimation and Valuation via Planning

As illustrated, for example, by the extensive literature on business-IT alignment (see, e.g., Luftman et al., 1999), systems are typically developed to enable a goal or a set of goals that are important for individuals, organizations, or societies. Goals for organizational systems, particularly those developed for the enterprise level, are typically expected to be aligned with the overall goals of the organizational unit(s) the system serves. When a software system is developed to serve an organizational goal or a set of organizational goals, the organization funding the development of the system is typically interested in ensuring a sufficient return on investment. This involves estimating reasonably accurately in advance how much the development of the system is going to cost. Further, it requires an estimation of the value of the business benefits that will be realized.

While acutely important in enterprise-level projects, this is true even in highly agile projects, where fixed time and fixed budget are nominally linked to a flexible definition of scope. Even in these cases, however, for systems targeted for production use, a certain minimum level of functionality – the "Minimum Viable Product" or MVP – needs to be delivered for the system to deliver meaningful value (Rubin, 2013, p. 295). In contrast, projects that are executed to develop a proof of concept or in some other way to explore the technical feasibility of a possible solution are by definition highly uncertain in terms of their outcomes. But even with these projects, it is important to articulate in advance the exploratory purpose of the work and the amount of time and talent the organization is willing to invest in that exploration.

Thus, all projects require a degree of estimating both costs and benefits. Further, it is impossible to accurately estimate the costs of developing systems capabilities that are not specified at all or that are specified only at a very high level of abstraction (McConnell, 2006, pp. 35-7). For agile projects, the implication of this point is that there should be sufficient time and budget to make it likely that (at least) the MVP can be delivered. Further, the only way to ensure this is to engage in more, on-

going refinement of requirements, designs, and estimations than is typically specified in "pure" agile approaches based on an early evaluation of short expressions of features, such as a product backlog consisting of user stories and value rankings (Cockburn, 2001, p. 187).

We argue, therefore, that a process to determine an appropriate degree of planning is a necessary part of all systems development projects. The extent to which it is formally performed and documented will be a matter of degree, but without any planning, no project is likely to succeed.

To reiterate, agile approaches tend to give short shrift to the level of formal requirements analysis needed to generate accurate estimates. Further, many existing SA&D courses and textbooks appear to consider questions related to software cost estimation and the relationship between planning and estimation too advanced to consider, particularly in introductory courses. This is unfortunate because without understanding the relationship between planning and estimation, our students will have difficulties understanding one of the key factors affecting the success of their systems projects. We, therefore, advocate teaching an enterprise-level approach that explicitly considers issues of estimation, including at multiple points during the systems project.

2.4 Understanding the Need for Requirements Analysis and Design Artifacts

From the above discussion, it follows that requirements specification (articulation of system capabilities before they are constructed) is a necessary part of all systems development projects. Again, the extent to which requirements are *formally* documented in detail varies depending on the project, but the functional and technical artifacts that enable the achievement of a beneficial human goal via a system do not appear randomly without the articulation of the key ideas of the system's role in advance. It is, however, possible and increasingly common (in agile projects) that requirements are not formally documented in detail (or at all, other than in the form of brief user stories) and that they literally may be specified informally and in real-time right before construction. This approach, typical of agile techniques, is often called "emergent requirements" (Boehm and Turner, 2004, p. 29), as opposed to the "Big Design Up Front" (Boehm and Turner, 2004, p. 55) typically associated with plan-driven approaches.

Furthermore, all systems have an internal technical design. The quality of the design and the extent to which the design is preconceived and formally documented before technical implementation varies significantly, but, whether the design is planned or emergent, it exists.

Projects exist for which highly emergent requirements and design are the best or at least perfectly reasonable choices. It is, however, essential for students to learn to understand the factors determining the degree to which a project should utilize formal, documented specification of requirements and design choices as well as anticipating the consequences of not engaging in formal specifications.

For example, it is essential that our students understand the relationship between the agility of the project approach, the use of formal requirements and design artifacts, and the role of the scope concept: plan-driven projects utilizing a traditional SDLC (e.g., "waterfall") have a fixed scope based on pre-specified formal requirements, agile projects have moderately emergent requirements and flexible scope (beyond a core minimum viable product), and hybrid projects have a semi-flexible scope based on "guardrails" type requirement specifications that specify the minimum viable product and also a maximum scope bounded by explicit "won't have" specifications with room for requirements changes during construction within those guardrails.

Beyond teaching only agile emergent requirements techniques (user stories supplemented with "barely sufficient," informal models), we believe it is essential for business/systems analysts to be able to understand this range of scope definition approaches, including the selection of an optimal scoping approach for any given project. This includes developing formal modeling skills to document the key concepts of the domain of practice for which the system is developed, the relationships between these concepts, and their relevant characteristics. Business/systems analysts also need to understand the organizational processes that the system under construction is expected to enable. The understanding of the target systems and organizational transformation developed through domain/conceptual data modeling (Topi and Ramesh, 2002) and business process modeling (Rosemann and vom Brocke, 2015) forms an essential foundation that would be useful for all systems development initiatives.

Further, as described below, students need to understand the appropriate level of requirements modeling at various stages of a project, including a preliminary level of modeling needed to execute a "buy versus build" decision, per the discussion above, leading to more extensive modeling (including, especially, technical design) in circumstances where a decision has been made to proceed with internal development.

With this toolkit of scoping approaches, requirements techniques, and timing principles in hand, students will be equipped with the skills that they will need for a broad range of systems projects, ranging from the enterprise level to the level where highly agile approaches are appropriate.

2.5 Need for a Broad Range of Competencies

The proposed framework and guidance regarding its use demonstrate that developing software-based systems requires a broad range of human competencies. Broadly speaking, these competencies can be divided into systems/business analysis (focused on understanding what goals the system should enable its users and user organization to achieve and what the functional system characteristics should be to enable those goals), sourcing of software components (essentially buy versus build), systems construction (focused on creation and modification of technical artifacts), configuration (focused on aligning the flexible capabilities of third party software to an organization's needs), and project management (focused on choosing and executing a project approach). These competency categories are different, and the number of individuals who exhibit all these types of competencies at a high level of achievement is small. Some methods (specifically agile ones) expect individuals involved in the development process to possess a general, minimum level of competence in all major skillset areas (Rubin, 2013, pp. 201-203), but this is unlikely to be feasible with large teams engaged in enterprise-level software development projects (Boehm and Turner, 2004, pp. 46-49). It is essential for students to understand how the competencies they acquire in an SA&D course fit in a project

as a whole and how their competencies are aligned with different approaches to systems development.

With respect to software construction, systems development projects often bring together human resources from a variety of sources, including internal salaried employees, long-term contractors, consulting organizations, etc. In enterprise-level projects, development resources are in many cases not all co-located. Any generalized framework needs to address these team characteristics, and any SA&D course needs to help students understand the variety of demands and mechanisms of support that different methodologies create for personnel working on the projects. To illustrate briefly, consider a project with a relatively small, cohesive, and talented on-shore team. Further, assume that the team had been delivering software successfully in a relatively agile fashion from a requirements perspective. Now, if management decides to augment or replace these team members with a new, offshore group of developers, then all other things being equal it is likely that there would need to be a shift toward a more plan-driven requirements approach to compensate for the loss of ability to establish and communicate requirements informally.

2.6 Enabling Planned Organizational Change

Particularly when SA&D courses are built around the agile principles, SA&D courses and current textbooks frequently omit the important discussion on the importance of preparing the user organization to successfully implement the changes that the deployment of the new capabilities require. Emergent requirements, iterative construction, and continuous testing do not necessarily mean that it is reasonable or even possible to assume continuous deployment of the constructed software capabilities to the target organization, per DevOps principles (Gruver and Mouser, 2015, p. 52). Delivering software capabilities is only one component in the change process that enables successful organizational transformation.

Rather, equally important are changes to the organizational processes that new software capabilities may enable but that will not be complete without changes in human behavior and organizational preparation. These typically include user acceptance testing, data cleansing and preparation, updated organizational policies and procedures, and training. It is frequently the case that organizational processes cannot, in practice, be changing continuously or in an unpredictable way. Instead, the degree of organizational change required needs to be proactively planned, managed, and supported. Therefore, even in agile projects, the deployment of new systems capabilities needs to be carefully evaluated and ultimately structured into planned releases that are completed much less frequently than DevOps-oriented construction iterations (e.g., major production releases twice a year and minor releases every three months). Carefully planned releases make it possible to prepare the organization for change through training, data preparation, changes in formal organizational roles, negotiations regarding contracts particularly in unionized environments, etc. It is essential that SA&D courses emphasize the importance of this preparation as part of the overall systems project process.

3. OVERALL STRUCTURE OF THE FRAMEWORK

Having introduced the areas that we believe will need additional focus compared to the existing typical practice in SA&D courses – including those emphasizing agile approaches – we are specifically proposing a highly generalized development approach framework that addresses the following key needs:

- Development projects need to be based on an articulation of organizational goals and the organizational target/future state that the target system needs to enable. The goals set for the organizational change enabled by the system should be understood by all actors in the development process. Detailed project planning is not possible until the core business transformation needs of the client organization have been articulated. Even highly agile organizations generally need an articulated goal for the change initiative enabled by the project. This future state analysis is typically supported by current state analysis, which fundamentally helps define incremental scope by contrasting the future visions with existing capabilities and may help the project find inconsistencies and technical problems that the organization was not aware of earlier.
- Increasingly, many systems development projects use systems and systems components acquired from external sources as components of the target system. These may include commercial-off-the-shelf (COTS) products, open source projects, components, and so on. Planning a development project is not possible without a solid understanding of the role these external systems and systems components play in the process. This, in turn, depends on developing a solid understanding of requirements to determine which components of the solution should be externally versus internally sourced. Equally important is to understand the sources and costs of human resources that will need to be used in project implementation from the perspective of configuration versus development. For example, a project emphasizing the configuration of third-party software would tend to need relatively more business analysts and fewer developers.
- Despite the recognized strengths of agile approaches to systems development, they still suffer from the key challenge that the underlying philosophy of promoting flexible scope, when fully implemented, may conflict with ensuring that a project should be able to deliver a minimum fixed scope (minimum viable product, or MVP, or minimum viable solution, or MVS) in conjunction with a fixed budget and fixed schedule. Further, the need to manage customer expectations by explicitly limiting maximum scope is also often neglected. In case a combination of minimum fixed scope, budget, and schedule is required, an organization needs to be able to make methodological choices that allow this. Students taking an SA&D course need to learn to understand the close linkage between development approach options and the relationship between scope, budget, and schedule.

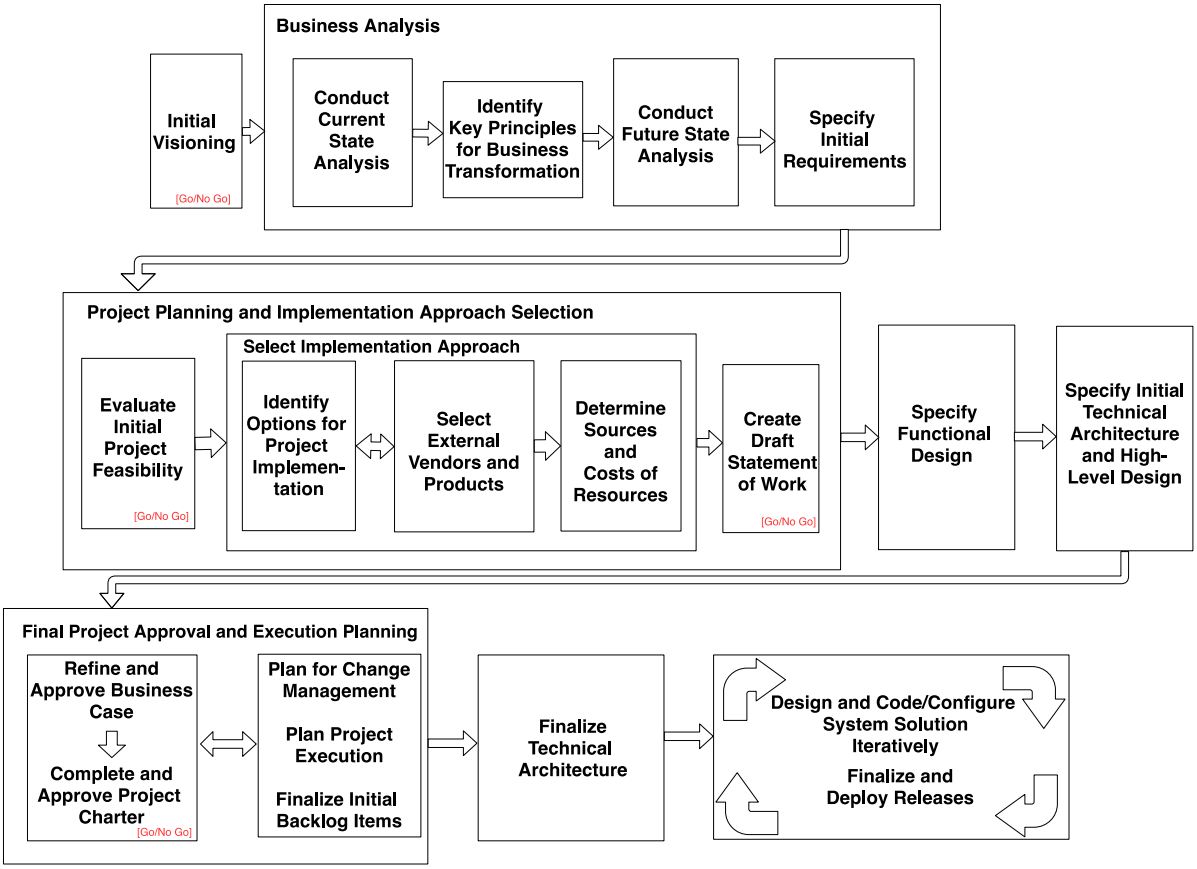


Figure 1. Proposed Development Process Framework

- The target organization adopting new systems capabilities needs to implement them effectively using change management techniques. This needs to be considered carefully when planning the mechanisms through which the new software is introduced to the organization. The software construction schedule should not dictate the system release schedule.
- Provide multiple opportunities to assess the return on investment, either explicitly in financial terms comparing development costs versus business benefits or else conceptually in general terms of supporting the organization's mission. This will provide an ability to make "go/no-go" decisions as the project progresses and costs and benefits estimates are progressively refined for greater accuracy.

Given the principles specified in the previous section and the needs articulated above, we propose a framework depicted in Figure 1 which consists of the key, high-level activities described below. Note that a) these are intentionally not called stages to avoid the impression that they are all required and always executed in the same order and b) the size of the graphical elements does not reflect the relative length of the activities – for example, an agile project with emerging requirements could consist almost entirely of the iterative construction activity.

3.1 Initial Visioning

Within this activity, the idea of the project is identified and articulated in the form of an initial vision (typically a brief narrative document articulating the key business problem or opportunity to be addressed, potentially paired with early, high abstraction-level visual models describing system requirements and associated anticipated business benefits). Based on this initial articulation of the project characteristics and associated screening-level cost and benefit estimates, the project is either deemed to be sufficiently important to justify immediate further exploration, set aside to wait, or abandoned. It is also possible that a project is found to be interesting but not at a level that would justify enterprise-level attention and resources; in these cases, the idea could be explored further in a Proof of Concept type of project. The nature of the project as envisioned within this activity contributes also to the initial determination of the most appropriate project approach on the spectrum from plan-driven to agile.

From the pedagogical perspective, a discussion regarding initial visioning will allow the students to learn about multiple potential sources of project ideas; the concepts of incremental, gradually increasing funding commitments and go/no-go decisions; high-level articulation of project goals and implementation ideas; and the alignment between project approach and other key characteristics of the project.

3.2 Business Analysis

In the proposed model, the business analysis activity consists of four primary sub-activities: 1) analysis and articulation of the **current state** characteristics of the target area (at least from domain and process modeling perspectives); 2) identification and articulation of the key changes that the system should enable within the target area; 3) analysis and articulation of the **future state** characteristics of the target area (again, both from domain and process modeling perspectives); and 4) the initial **system requirements specification** with high-level user stories and user interface prototypes. The importance of current state analysis and future state analysis may be established by noting that initial requirements – capabilities that do not yet exist and therefore need to be built or acquired – are generated by conceptually subtracting the current state capabilities from the overall future state capabilities. By the time of completion of the business analysis activity, the organization and the development team have developed a much better understanding of the domain of interest, its key business processes, the relevant organizational actors within the domain, and the ways in which the system to be developed will enable the various organizational roles to achieve their goals. The goals are articulated with user stories that form the core requirements specification for the system of interest.

Importantly, note that this level of requirements analysis is not a *direct* predecessor to systems construction. Rather, it is preparation for key project planning activities, as described in the next section. Naturally, the results of this activity will later guide functional design (if conducted) and construction. Any decisions made by this time regarding the project approach will have an impact on the level of details of business analysis. For example, if the organization has decided to use an agile approach to systems development, it is likely that the project will expend less time and focus on requirements specification given the agile projects' focus on emergent requirements.

Pedagogically, this activity enables the coverage of a number of essential concepts and skills, including:

- Process modeling: typically with the Unified Modeling Language (UML) activity diagram or Business Process Modeling Notation (BPMN)
- Domain modeling or conceptual data modeling: typically with the UML class diagram grammar or Extended Entity-Relationship diagram (EER)
- Deriving epics and user stories
- Early-stage, low-fidelity user interface models: with wireframes, mock-ups, or prototypes (but not yet for UI design purposes; the intent is to provide customers with illustrations that are easier to understand than textual descriptions).

3.3 Project Planning and Implementation Approach Selection

The results of the business analysis activity form a foundation for project planning, with the assumption that the systems development project is driven by the desired changes in the human activity of interest articulated in the future state analysis. Project planning is divided into multiple sub-activities in the model, including the following:

1. Before moving forward with the implementation approach selection, the first needed action is to evaluate the project's **initial feasibility** based on the results of business analysis. At this point in the process, it should be possible to create an early stage economic analysis, determine whether or not the project is technically possible to implement, analyze the implications of a reasonable project schedule, consider the legal and political implications of the project, and establish the availability of funding, all typical elements of project feasibility. Again, a go/no-go decision process will be considered. It is possible that at this point the results of the analysis suggests that the project should not continue because it is likely that its implementation is not feasible, leading to discontinuation of the project.
2. **Selecting key characteristics of the implementation approach**, including buy versus build and sourcing of implementation resources. Increasingly, the chosen approach integrates system components from multiple sources and uses many different types and sources of resources. Thus, buy versus build is not, in practice, merely a decision between internal development and procuring an external software package; instead, the analysis might lead to the conclusion that the ultimate systems solution will utilize elements from commercial-off-the-shelf (COTS) and/or open source offerings integrated with or augmented by internal development. Components acquired from external sources may be available as a cloud-based solution or provisioned on-premises, or both. Similarly, the model chosen for sourcing of resources may lead to a complex integration of salaried employees, individual contractors, and contract employees from consulting firms.
3. Assuming the previous analysis suggests the use of one or more externally sourced products or components, then a **selection process may be needed to evaluate and select those external possibilities**. This, in itself, may amount to a small project, including soliciting information from vendors or open source project groups, for example: requests for proposal (RFPs), requests for information (RFIs), demonstrations, sandbox implementations, Proof-of-Concept integrations, pricing negotiations, and so on. More generally, the mix of externally and internally sourced capabilities will provide the basis for overall resourcing decisions: external software code, software construction labor resources, software configuration labor resources, and so on.
4. Based on decisions regarding the characteristics and sources of a project's software components, sourcing of the project's creative resources, and the development approach, creating a draft set of key project documents, including a refined articulation of project vision incorporated into a **statement of work** and an **analysis of business benefits and systems costs**, building on the findings of business analysis. Also at this point, it will be possible to make further decisions regarding the optimal development approach. As discussed above, some projects justify the use of a highly agile approach, others a hybrid approach incorporating significant up-

front requirements analysis and functional designs with iterative construction, and still others (although less frequently) a highly plan-driven (traditional SDLC) approach. This complex decision should be evaluated based on all of the factors shown in Table 1. Based on all this work, another go/no-go decision is made to either move on with the project or discontinue it.

Pedagogically this activity will introduce a number of highly important concepts and skills: a more detailed understanding of key project elements (feasibility analysis, statement of work, and early cost-benefit modeling); a rich description of the options available in the buy versus build decisions; a recognition that creative resources for software development, configuration, and deployment come from a variety of sources, which have to be carefully selected; an in-depth understanding of the factors that impact the selection of an appropriate development approach; a process for selecting a specific approach; and an enforcement of the concept of increasingly detailed cost-benefit analyses.

3.4 Functional Design Specification

Based on the results of the project planning and implementation approach selection activity, the project may or may not decide to engage in detailed functional design specifications, depending on the selected methodological approach.

The functional design process specifies in a manner visible and meaningful to business users the details of how the system will solve the business problem. The deliverables of functional design are clearly distinct from technical design artifacts such as design class diagrams and sequence diagrams, which are typically only utilized by IT staff members.

If the organization has decided to use a highly agile approach for the construction phase of the project, functional design specifications are not necessary because agile requirements specification is emergent and takes place right before construction based on the initial ideas generated by user stories. Furthermore, the extent to which certain functional design documents are needed also depends on the software solution sourcing approach. For example, in the case of a project that is entirely based on a SaaS or COTS solution, functional design specification in the form of use cases or detailed UI/UX prototypes might be much less important than in a project in which software is developed largely from scratch.

When conducted, functional design uses output from business analysis as its foundation: the purpose of the systems solution is, after all, to enable the business transformation envisioned in business analysis. In addition, the business analysts responsible for the creation of the functional specification conduct further requirements discovery and structuring activities. The primary deliverables from this activity for internal development or integration code include use cases and/or use case slices. In this context, "slices" refer to refinements of use cases intended to groom the backlog so that it can be constructed in a series of non-overlapping sprints and consisting of a single software technology per slice (Jacobson, Spence, and Kerr, 2016). Cockburn (2001, p. 169) refers to a similar concept with the term use case "splitting." User interface mock-ups or prototypes are also used to clarify the specification and make it more concrete, especially for capabilities developed internally. The mock-ups and prototypes developed as part of the functional design are significantly more detailed than those created earlier as part of business analysis.

In general, at the enterprise level, the functional design artifacts will be completed ahead of the construction sprints, although the exact degree and sequencing of up-front planning will differ from project to project, depending on the earlier decisions regarding the level of agility of the project: some projects engage in significant "Big Requirements Up-Front," others may elaborate requirements in "shadow sprints" one to two iterations ahead of construction, while in others, the requirements may be fully emergent, determined right before or concurrently with construction without any pre-specification. The choice between plan-driven and agile is not binary but varies on a scale. For example, Serrador and Pinto (2015) used the extent to which the project's overall requirements planning was conducted prior to starting software construction as one of the measures of the agility of the project.

Pedagogically, the functional design activity addresses an important set of concepts and skills, particularly those related to methods of specifying requirements at a detailed level: detailed use case narratives (fully dressed user goal level use cases in Cockburn's (2001) terminology) and detailed user interface mock-ups and prototypes. It is also possible to introduce system sequence diagrams here to provide additional specificity for the fully dressed use cases. Moreover, the activity introduces the students to the question regarding the extent to which detailed functional design is necessary, depending on earlier decisions regarding the nature of the development approach.

3.5 Specifying Initial Technical Architecture and Technical Design

This activity recognizes the potential role of the development of initial technical architecture and high-level technical design as a foundation for more refined budgeting and, ultimately in a more detailed form, architecture and design for iterative construction.

Within this activity, the project addresses key questions regarding the provision of data storage, processing, and communication capabilities. Furthermore, it identifies relevant organizational data resources, shared software component libraries, and details of user experience technologies, also finalizing the characteristics of the development and deployment stack used for the project. Pedagogically, this activity is an important opportunity to demonstrate to the students the impact of technical architectural choices on the business case for the project.

In some cases, this activity does not require any additional work, for example when a systems project extends an effective, existing enterprise architecture within the organization. Still, it is important to at least evaluate possible revisions or extensions to a technical architecture.

3.6 Final Project Approval and Execution Planning

If the selected development approach requires detailed functional design, that activity will make a significant contribution to the development of the final project plan for software construction: more detailed specifications will enable more accurate estimation and initial sizing of the construction backlog items. The need for this kind of planning is a hallmark of enterprise-level projects, where large budgets in absolute

terms tend to require more documented work to obtain needed budget approvals. Further, this is needed to combat the specter of the "planning fallacy," which is the well-documented risk of systematically underestimating project costs – this risk is magnified at the large scales of enterprise-level projects (Shmueli, Pliskin, and Fink, 2016).

More accurate estimation will, in turn, be an important contribution to a refined business case. In small-scale, fully agile projects that skip the functional design activity, there might not be a need (or any additional information) for any further action related to the business case. In either case, at this stage, it is time for the final approval of the project's budget via a business case and the development and approval of a project charter.

Once the project has been approved, it can move forward to an appropriate level of revised execution planning, including the preparation for construction, configuration, and the organizational change that is always part of any substantial systems development project. In execution planning, particularly for enterprise-level systems, it is likely that it will be necessary to map stories to sprints in advance to ensure the likelihood of delivering an MVP capability in the time and budget contemplated. The nature of iterative construction means, in practice, that those mappings will be revised as the project progresses, which is acceptable as long as the guardrails specified in hybrid planning are not violated. The intent is to develop backlog items sized so that they can be implemented in a single sprint, based on the planned length of the iterations and an understanding of the development team's capabilities.

The finalization of the technical execution plan and the change management plan may lead to an improved cost-benefit analysis and have an impact on the budget.

One of the most difficult dimensions of an enterprise-level SA&D project to incorporate successfully into an educational experience is the actual organizational deployment of systems capabilities (including planning and preparation for the deployment). As discussed above, it is essential that the students understand the essential role of preparing the target organization for the change that successful software-enabled organizational transformation requires. The framework includes planning and preparation of the organization for deployment for all types of projects, including the highly agile ones.

3.7 Finalizing Technical Architecture

Execution planning discussed above may reveal characteristics of the project that require additional refinement of the technical architecture initially specified before final project approval. Regardless of the chosen project approach, it is beneficial if design and construction are based on well-defined architectural choices for elements such as data management, software design, inter- and intra-system communication, use of hardware at premises or cloud-based solutions, services and components, and systems security. Pedagogically, this activity offers an excellent opportunity to demonstrate how various elements of system architecture enable effective design and construction.

3.8 Iterative Design, Construction, and Configuration

The proposed process framework assumes that the construction and configuration of deployable systems capabilities will take place using an iterative process. The specific features of construction vary somewhat depending on the overall process characteristics: If the overall process approach is highly agile, construction is naturally integrated into the overall iterative structure together with emergent requirements and design. In hybrid projects, however, at least functional requirements and often also functional design are specified before construction starts (although, as noted earlier, a degree of revision within the "guardrails" scope is possible during construction). Unlike highly plan-driven processes, hybrid processes do not include one long, monolithic construction stage that leads to a single, potentially very large deliverable. Thus, most aspects of Scrum are maintained (excluding emergent requirements) in hybrid processes: fixed-length, short iterations; self-organizing teams; active role of a product owner; initial planning, brief daily team meetings, and a review and retrospective at the end of each iteration. The main difference between agile and hybrid approaches is that in hybrid processes, the initial project backlog has already been elaborated into significant functional requirements details, allocated to iterations according to project-level plan, and scheduled for the duration of the project. This, in turn, allows the specification of expected project completion time and an upper limit of project costs for a specific minimum set of system capabilities.

Compared to hybrid, agile processes do exhibit some clear differences: in them, project teams make decisions regarding the backlog items they select for each of the iterations, requirements are not pre-specified but emerge from communication between the developers and the product owner, and there are no specific commitments regarding a budget, scope, and schedule combination. Put more succinctly, agile utilizes fixed budget and time, but flexible scope beyond MVP. In contrast, hybrid utilizes fixed budget and time, but semi-flexible scope guaranteeing a minimum viable product also constrained by a maximum scope limit (i.e., explicitly excluding "Won't Have" features). Pure plan-driven processes (with a single, lengthy construction and testing stage) are quite rare today, as the benefits of receiving and incorporating frequent customer feedback after iterations are manifest. Thus, we recommend that in SA&D education contexts the focus should be on agile and hybrid approaches.

In pedagogical contexts, it is important to note and emphasize that iterative construction may consist of a rich variety of activities depending on the nature of the project. In some projects, most of the work consists of configuration (of COTS and SaaS solutions); in others, externally developed components are integrated together ("glue code") with internally developed modules and extended user interfaces ("surround code"); and in others all software capabilities required in a project are constructed fully from scratch.

We believe that SA&D course projects that are sufficiently long to enable iterative construction combined with requirements specification and functional design are clearly more effective in helping students understand the end-to-end complexities of a systems development project compared with course projects that are limited to only requirements specification or only iterative construction. One possible way to enable iterative construction in a semester-long project is to use one of the low-code, high-abstraction level development platforms, such as Mendix or Salesforce. Such low-code platforms help overcome two barriers to engaging in software construction in an SA&D course: first, that many SA&D

students lack the low-level programming skills to create software and, second, that low-level programming languages require too much time to create reasonably complex and realistic software in the context of a one-semester SA&D course.

4. KEY BENEFITS OF THE PROPOSED FRAMEWORK

The proposed hybrid framework offers many advantages as a foundation for an SA&D course compared to primarily plan-driven and primarily agile approaches. These include the following:

- Recognizes formally the importance of determining an appropriate level of planning across all types of projects, including agile ones.
- Recognizes formally the general need to base requirements specification on business analysis, including domain modeling, current and future state business process and domain modeling, and early-stage user interaction prototypes/wireframes (the latter especially when internal development is utilized).
- Recognizes that in many contexts it is important to be able to execute projects with a fixed time, a fixed budget, and a certain minimum project scope necessary to deliver business value (the "MVP" or "Minimum Viable Product"), as well as a maximum scope definition in some circumstances.
- Supports the determination of internal versus external sourcing of software, including the use of the same overall model for projects based on internal development and projects that focus on the configuration of system components procured from external vendors (COTS, SaaS, etc.).
- With respect to the previous point, the framework calls for extending the base requirements specification to detailed functional designs only in circumstances where that is appropriate (i.e., when the software will be built from scratch rather than sourced from external products or components).
- Supports appropriate go/no-go decisions at multiple points in the project. Further, supports the more formal level of planning and project approvals that are typically required in enterprise-level projects, including expanded estimation that is appropriate to mitigate planning fallacy risks at the enterprise level.
- Addresses the need to evaluate, design, and/or revise technical architectures at multiple points during the project.
- Through iterative development, recognizes the importance of learning that takes place during the project execution and the changes that occur because of the changing environment and learnings resulting from iterative development itself.
- Maintains the demonstrated benefits of agile development and integrates them with the benefits of guidance from business analysis–based requirements specification.
- Provides a balance between the streamlined efficiencies of agile-style DevOps continuous integration and deployment innovations with the on-going need at the enterprise level to engage in needed change management planning, including addressing business policies and procedures, training, user testing, and data preparation.
- Finally and above all, provides students with a unified and adaptable overall SA&D toolkit, ranging from highly agile to highly plan-driven, that can be effectively tailored to work effectively from the smallest, simplest projects to large, complex, and highly consequential projects at the enterprise level.

5. CONCLUSION

In this paper, we have proposed a generalized systems development process framework for SA&D courses that is based on the key idea that different systems projects need different project activities and structures and different systems development approaches. The proposed framework supports enterprise-level projects fully, but it can also be selectively scaled back toward increased agility to effectively support smaller, less complex projects.

The framework addresses several aspects of systems development projects and SA&D education that most existing frameworks and approaches either ignore entirely or discuss only in a cursory way. The most important ones of these include the following:

1. The framework specifically and intentionally addresses the role of various commonly used external sources of software capabilities, including COTS, SaaS, open source components, and their integration with internally developed components.
2. The framework recognizes the essential but varying role of planning at different levels of systems development. We specifically suggest that every project should include certain planning activities to ensure that the development project itself is conducted using the most efficient and effective approach.
3. The framework takes explicitly into account the inherent integration between the outcomes of systems development and the change processes that deployment of systems capabilities into the organization enables and/or requires. It also explicitly addresses the need for planning for organizational change management as an essential component of systems deployment.

In general, the framework creates a unified pedagogical foundation for teaching SA&D via a comprehensive, highly adaptable framework combining the strengths of both agile and plan-driven approaches.

We hope that this framework will be useful by providing foundational and architectural support for SA&D courses so that it will be easier for faculty members to recognize the diverse modeling and project management needs of different types of systems development initiatives.

This paper itself is a step in a lengthy process through which we are introducing these ideas in an integrated form for feedback and consideration to the academic and practitioner communities. We hope it will initiate active discussion among

those who teach system analysis and design and design curricula for IS programs.

6. ACKNOWLEDGEMENTS

An early, materially different version of this article was published in the proceedings of the 2018 SIGSAND Symposium. We gratefully acknowledge the valuable feedback from SIGSAND reviewers and conference attendees.

7. REFERENCES

Ambler, S. & Lines, M. (2012). *Disciplined Agile Delivery*. Indianapolis, IN: IBM Press.

Boehm, B. & Turner, R. (2004). *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston, MA: Pearson Education.

Chuang, S., Luor, T., & Lu, H. (2014). Assessment of Institutions, Scholars, and Contributions on Agile Software Development (2001-2012). *The Journal of Systems and Software*, 93, 84-101.

Cockburn, A. (2001). *Writing Effective Use Cases*. Upper Saddle River, NJ: Pearson Education.

Couger, J. D. (1973). Curriculum Recommendations for Undergraduate Programs in Information Systems. *Communications of the ACM*, 16(12), 727–749.

Dybå, T. & Dingsøyr, T. (2008). Empirical Studies of Agile Software Development. *Information and Software Technology*, 50(9-10), 833-859.

Fowler, M. (2003). *Patterns of Enterprise Application Architecture*. Upper Saddle River, NJ: Pearson Education.

Gruver, G. & Mouser, T. (2015). *Leading the Transformation: Applying Agile and DevOps Principles at Scale*. Portland, OR: IT Revolution.

Jacobson, I., Spence, I., & Kerr, B. (2016). Use-Case 2.0. *Queue*, 14(1), 94-123.

Knaster, R. & Leffingwell, D. (2018). *SAFe 4.5 Distilled: Applying the Scaled Agile Framework for Lean Enterprises.* Boston, MA: Addison-Wesley Professional.

Larman, C. & Vodde, B. (2017). *Large-Scale Scrum: More with LeSS.* Boston, MA: Pearson Education.

Leffingwell, D. (2007) *Scaling Software Agility: Best Practices for Large Enterprises*. Boston, MA: Pearson Education.

Luftman, J., Papp, R., & Brier, T. (1999). Enablers and Inhibitors of Business-IT Alignment. *Communications of the Association for Information Systems*, 1(3), Article 11.

McConnell, S. (2006). *Software Estimation: Demystifying the Black Art*. Redmond, WA: Microsoft Press.

Nunamaker, J. F., Couger, J. D., & Davis, G. B. (1982). Information Systems Curriculum Recommendations for the 80s: Undergraduate and Graduate Programs. *Communications of the ACM*, *25*(11), 781–805.

Petersen, K., Badampudi, D., Shah, S. M. A., Wnuk, K., Gorschek, T., Papatheocharous, E., Axelsson, J., Sentilles, S., Crnkovic, I., & Cicchetti, A. (2018). Choosing Component Origins for Software Intensive Systems: In-House, COTS, OSS or Outsourcing? – A Case Survey. *IEEE Transactions on Software Engineering*, 44(3), 237-261.

Rosemann, M. & vom Brocke, J. (2015) The Six Core Elements of Business Process Management. In *Handbook on Business Process Management 1,* 105-122, Springer Berlin Heidelberg.

Rubin, K.S. (2013) *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Upper Saddle River, NJ: Pearson Education.

Serrador, P. & Pinto, J. (2015). Does Agile Work? — A Quantitative Analysis of Agile Project Success. *International Journal of Project Management*, 33(5), 1040-1051.

Shmueli, O., Pliskin, N., & Fink, L. (2016) Can the Outside-view Approach Improve Planning Decisions in Software Development Projects? *Information Systems Journal*, 26(4), 395-418.

Spurrier, G. & Topi, H. (2017). When is Agile Appropriate for Enterprise Software Development? *Proceedings of the Twenty-Fifth European Conference on Information Systems (ECIS)*, Guimarães, Portugal.

Topi, H. & Ramesh, V. (2002). Human Factors Research on Data Modeling: A Review of Prior Research, an Extended Framework and Future Research Directions. *Journal of Database Management*, 13(2), 3-19.

Topi, H., Karsten, H., Brown, S. A., Carvalho, J. A., Donnellan, B., Shen, J., Tan, B. C. Y., & Thouin, M. F. (2017). MSIS 2016: Global Competency Model for Graduate Degree Programs in Information Systems. *Communications of the Association for Information Systems*, 40(1).

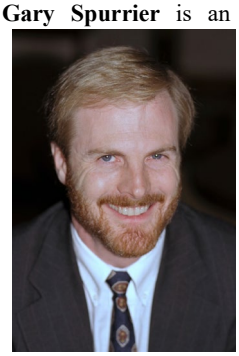
Topi, H., Valacich, J. S., Wright, R. T., Kaiser, K., Nunamaker, J. F., Sipior, J. C., & de Vreede, G. J. (2010). IS 2010: Curriculum Guidelines for Undergraduate Degree Programs in Information Systems. *Communications of the Association for Information Systems*, *26*(18).

Turk, D., France, R., & Rumpe, B. (2005). Assumptions Underlying Agile Software Development Processes. *Journal of Database Management*, 16(4), 62–87.

West, D., Gilpin, M., Grant, T., & Anderson, A. (2011). Water-Scrum-Fall is the Reality of Agile for Most Organizations Today. *Forrester Research*. Retrieved from https://www.forrester.com/report/WaterScrumFall+Is+The+Reality+Of+Agile+For+Most+Organizations+Today/-/E-RES60109.

AUTHOR BIOGRAPHIES

Heikki Topi is a professor of computer information systems at Bentley University. His Ph.D. in management information systems is from Indiana University. His research focuses on systems development methodologies, information systems education, and human factors and usability in the context of enterprise systems. His scholarly output includes journal articles, conference papers, large-scale edited volumes, textbooks, and curriculum recommendations (including *IS2010* and *MSIS2016*, the latest IS curriculum revisions as task force co-chair). He is currently Vice President of Education for the Association for Information Systems.



Gary Spurrier is an assistant professor of management information systems at the University of Alabama. He has held many industry positions in information technology, including CIO, COO, project leader for enterprise-level software projects, commercial off the shelf (COTS) software product manager, and IT and operations consultant. He earned his Ph.D. in MIS at Indiana University. His research focuses on enterprise software development and systems analysis and design.